Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Image and Signal Processing (CIISP 2007)

Function Approximation Through Growing Neural Network Based On RBF And Potential Functions

Iren Valova
Computer Science
University of Massachusetts Dartmouth
285 Old Westport Rd.
N. Dartmouth, MA 02747
ivalova@umassd.edu

Natacha Gueorguieva
Computer Science
City University of New York
2800 Victory Blvd.,
Staten Island, NY 10314
natachag@csi.cuny.edu

George Georgiev
Computer Science
University of Wisconsin Oshkosh
800 Algoma Blvd.,
Oshkosh, WI 54901
georgiev@uwosh.edu

Abstract - **This research work proposes a neural network based on RBFs with symmetrical potential functions and two fundamental components - potential function generators (PFG) and potential function entities (PFE). The approach, based on RBFNs with symmetrical potential functions (SPF), performs a mapping based on a set of generated potential fields over the domain of input space by a number of potential function entities. The placement and parameterization of the local units as well as the choice of their number is difficult and critical part for RBFs Networks. Networks with too many parameters can overfit data and have poor generalization. The presented method allows effective determination of all these values automatically. The proposed approach is suitable for on-line and off-line applications.**

I. Introduction

Approximating multidimensional functions by neural networks (NN) is a convenient way of representation because learning in NN corresponds to the approximation of an underlying function due to the built-in capacity to adapt synaptic weights to changes in the surrounding environment. The study represented in [1] concluded that "The crucial factor for a successful approximate algorithm is the choice of the parametric approximation architecture(s) and the choice of the projection (parameter adjustment) method." For neural networks the same can be expressed in the following way: The two most important factors for a successful NN approximating algorithm are the choice of the neural network topology and the effectiveness of the learning algorithm.

NN are universal approximators that can learn data by example [2] or reinforcement [3], either in batch or sequential mode. Most of reinforcement-learning algorithms consist in evaluating a value function that estimates the outcome of acting from a particular state. When the system to be controlled can be in a very large number of states, this value function has to be estimated by a generalizing function approximator. For small scale problems, the value function can be represented as a table. However, using function approximators requires making crucial representational decisions (e.g. the number of hidden units and initial weights of a neural network). Poor design choices can result in estimates that diverge from the optimal value function and agents that perform poorly.

However, solving large scale problems additionally requires developing of temporal difference methods (TD) which combine principles of dynamic programming with statistical sampling, usage of the immediate rewards received by the agent to incrementally improve both the agent's policy and the estimated value function for that policy.

An artificial neuron can be linear (implementing linear parameterizations, which are characterized by weighted combinations of basis functions) or nonlinear. Nonlinearity is a highly important property, particularly if the underlying physical mechanism responsible for generation of the input signal is nonlinear. NN can easily learn an input-output mapping of multidimensional nonlinear functions because of their parallel architecture thus making NN the perfect tool for certain tasks. Every neuron in the network is potentially affected by the global activity of all other neurons in the network. As univariate approximation theory does not generalize well to higher dimensional spaces [4], well-known parametric structures, such as splines and wavelets are convenient tools when dealing with input spaces with up to three dimensions [5]–[7]. For example, the majority of spline-based solutions for multivariate approximation problems are based on tensor product spaces that are highly dependent on the coordinate system of choice [8, 9].

NN with linear parameterization and least-squares approach using a linear combination of predetermined under-complete basis functions have shown promising results [10]. Common basis function choices for general function approximation problems include Fourier functions (trigonometric polynomials), Gaussian kernels [11], and wavelets [12]. Both Fourier bases (global functions) and Gaussian kernels (localized functions) have smoothness properties that make them particularly useful for modeling inherently smooth, continuous functions while wavelets provide basis functions at various different scales and may also be employed for approximating smooth functions with local discontinuities. The latter make the implementation of Fourier functions or Gaussian kernels as basis functions not always appropriate for the purposes of approximation. On the other hand wavelets posses over-complete bases, i.e. one has to appropriately choose a subset of basis functions which is not a straightforward task in practice and thus limits their implementation as well.

In recent years, multi-layer feedforward and radial basis functions neural networks have been widely used for pattern classification, function approximation and regression problems. The problem of determining the analytical description for a set of data arises in numerous applications, and neural networks are a convenient way of representation because they are universal approximators that can be trained to map multidimensional nonlinear functions.

Function approximation methods fall into two broad categories: global and local. Global approximations can be made with many different function representations, e.g. polynomials, rational approximation, multi-layer perceptrons, radial basis functions [13]. Often a single global model is inappropriate because it does not apply to the entire state space. To approximate a function $f(\boldsymbol{X})$, a model must be able to represent its many possible variations. If $f(\boldsymbol{X})$ is complicated, there is no guarantee that any given representation will approximate it well. The dependence on representation can be reduced using local approximation where the domain of $f(\boldsymbol{X})$ is broken into local neighborhoods and a separate model is used for each neighborhood. It has been shown that an MLP and RBFs neural network, with a single hidden layer, can approximate any given continuous function on any compact subset to any degree of accuracy, providing that a sufficient number of hidden layer neurons are used [14].

Function approximation problem with neural networks is formulated in the following way: from a given a set of training examples $\{[\boldsymbol{X}, f(\boldsymbol{X})]\}$ of an unknown function $f : \Re^n \to \Re$, we want to design a network that learns $\phi(\boldsymbol{X})$ which is a good approximation of $f(\boldsymbol{X})$. We are interested in a more general solution of this problem, where $f(\boldsymbol{X})$ is a function of several arguments represented by $\boldsymbol{X} = [x_0, x_1, ..., x_n]$.

Mathematically, a neural network can only evaluate a special function, depending upon its architecture. For example, if $n,\ s \geq 1$ are integers, the output of a neural network with one hidden layer containing n principal elements (neurons), each evaluating a nonlinear function ϕ, and receiving an input vector $\boldsymbol{X} \in \Re^s$ can be expressed in the form $\sum_{k=1}^{n} a_k \phi(\boldsymbol{w}_k\ \boldsymbol{x} + b_k)$, where for $k = 1,2,...,n$, the weights $\boldsymbol{w}_k \in \Re^s$, the thresholds b_k and the coefficients a_k are real numbers.

The universal approximation capability of multilayer feed-forward neural networks (FNN) widely studied by well known researchers revealed that if the network's activation functions comply with an explicit set of assumptions (which vary from one paper to another), then the network can indeed be shown to be a universal approximator [15]. The commonly used activation function in these papers are the sigmoid function and the generalized sigmoid function. One common characteristic of these activation functions is that they are fixed and therefore they can not be adjusted to adapt to different approximation problems [16, 17]. The neuron activation function is crucial as the performance of FNN depends mostly on it. It has been proven that FNN with a single hidden layer can uniformly approximate continuous functions if the activation function is locally Riemann integrable and nonpolynomial [18]. Authors in [19] proved that FNN with a locally bounded piecewise continuous activation functions can approximate any continuous functions to any degree of accuracy if and only if the activation function is not a polynomial.

II. Approximation with FNN and RBF

There are two still existing problems in function approximation with FNN. The first can be summarized as: if the function to be approximated is a piecewise continuous function which contains finite or infinite continuous parts, a continuous approximator can not solve problems such as nonlinear and continuous data simulation [20]. The FNN group topology proposed in [20] is a generalized neural network where each element is a separate neural network. Some additional operation as addition and product of any two elements are defined. However, if a piecewise continuous function to be approximated is made of infinite sections of continuous function, the neural network group will have to contain infinite number of neural networks, making the simulations very complicated because each continuous section has to be approximated independently and separately. The authors in [20] did not propose a learning algorithm for the neural network group what limits the applicability of the proposed topology. The second FNN problem which has to be solved requires emphasis on studies on setting some free parameters with the activation function in order to construct some kind of neuron-adaptive function. The latter will provide better fitting capabilities than the fixed activation function and developing a specific learning mechanism for the neural network group allowing using these parameters in effective way.

The radial basis function networks (RBFN) correspond to a particular class of function approximators, which can be trained by using a set of learning samples [21]. The strategy used in RBFNs consists of approximating an unknown function with a linear combination of non-linear functions, called radial basis functions [22, 23]. The latter may be chosen to be either local or global, and they may or may not incorporate shape parameters, which can be tuned to reflect the nature of the data. The locations of the basis functions may also be adapted using a variety of clustering techniques or via a nonlinear optimization.

A relevant property usually required for a class of approximators is the universal approximation. In general, an approximator is said to be universal if it is theoretically capable to approximate any integrable function to a reasonable degree of precision. In [24, 25] it was proven that RBFNs with only one layer of hidden units can uniformly approximate any continuous function to an arbitrary precision if enough units are provided [24], i.e. the accuracy improves with the number of hidden units.

The determination of the type of RBFs used for a given set of data is an active area of research. The standard RBF neural network with a single output neuron produces a mapping function $f : \boldsymbol{x} \rightarrow f(\boldsymbol{x})$, where the n-dimensional input vector $\boldsymbol{x}$ is submitted to the neural network and the scalar output $f(\boldsymbol{x})$ is obtained to construct the classification rule. The typical RBF neural network mapping has the following form

$$f(\boldsymbol{x},\boldsymbol{w},\boldsymbol{p}) = \sum_{i=1}^{N} w_i K_i(\|\boldsymbol{x}\|_i, \boldsymbol{p}_i) \tag{1}$$

where N is the number of the neurons in hidden layer, $K_i(\|\boldsymbol{x}\|_i, \boldsymbol{p}_i)$ is the *i*-th radial basis function, $\boldsymbol{p}_i$ is the vector of adjustable parameters (centers, biases etc.). The RBFNs representational ability combined with computational and analytical tractability comes from the linear combination (Eq.1) of typically nonlinear basis functions. Specifically, the radial nature of the functions derives from the choice of basis functions $K_i(\|\boldsymbol{x}\|_i, \boldsymbol{p}_i)$, where the output of each one depends only upon the distance of the input to another predetermined point $\boldsymbol{x}_i \in T_N$, where T_N is the training (learning) set.

The proper choice of the basis functions and their number is one of the main problems when designing an RBF network, i.e. a small number of functions may result in poor approximation accuracies, while a large one can lead to poor classification performance of the neural network. For a chosen number and type of network basis functions, two sets of parameters need to be determined (trained) in order to make the network perform correctly in the desired data processing task. First, there are the parameters on center positioning of the basis functions within the space spanned by the network input data. Second, there is a set of output weights, which propagate basis function responses to be linearly combined at the network output layer. Accordingly, the training task of a radial basis function network can be broken into two phases: selection of basis functions centers, followed by determination (training) of the output layer weights.

Commonly used RBF network assumes a Gaussian basis function for the hidden units

$$\varphi_i(\boldsymbol{x}) = exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{\mu}_i\|^2}{2\sigma_i^2}\right) \tag{2}$$

where $\boldsymbol{x}$ is the d-dimensional input vector with elements x_i, σ_i^2 is the standard deviation of the i-th unit receptive field and $\boldsymbol{\mu}_i = \|\mu_{ij}\|$ is the mean vector determining the center of basis functions φ_i. The RBF neural network output is determined by

$$f(\boldsymbol{x}) = \sum_{i=1}^{N}\left[a_i exp\left\{-\frac{\|\boldsymbol{x}-\boldsymbol{\mu}_i\|^2}{2r^2}\right\}\right] + b \tag{3}$$

where N is the number of the number of the hidden layer neuron, r is the RBF width, and a_i and b are the adjustable parameters.

III. Function approximation with potential functions

Given a set of training examples patterns $\{\ \boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\ \}$ of an unknown function $f : \mathfrak{R}^n \rightarrow \mathfrak{R}$, we want to receive the approximation $f^*(\boldsymbol{x})$ of $f(\boldsymbol{x})$ such that a certain performance criteria is optimized. In the simplest case when $f(x)$ is a function of a single scalar variable x, for some finite discrete values of argument x, this represents a curve fitting problem. We work with a more general version of this problem when f is a function of finite number of multidimensional patterns $\boldsymbol{X}$ represented by $\boldsymbol{X} = (x_0, x_1, ..., x_m)$. We assume that there is an existing system of basis functions $\phi_1(\boldsymbol{x}),\ \phi_2(\boldsymbol{x}),...,$ such that the approximation function $f^*(\boldsymbol{x})$ can be expressed as a linear combination of functions $\phi_1(\boldsymbol{x}),\ \phi_2(\boldsymbol{x}),...,$ in the form

$$f^*(\boldsymbol{x}) = \sum_{j=1}^{\infty} c_j^* \phi_j(\boldsymbol{x}) \tag{4}$$

if $f^*(\boldsymbol{x}) \in \Im_\phi$ and the mathematical expectation of function square where $M\{|f^*(\boldsymbol{x})|^2\} < \infty$ is finite. ($\Im_\phi$ is the functional space of basis functions).

Potential function approximation algorithms for the case of functions without a noise may be expressed as

$$\begin{gathered} f_{n+1}(\boldsymbol{x}) = f_n(\boldsymbol{x}) + \gamma_{n+1}\boldsymbol{r} \\ [f(\boldsymbol{x}_{n+1}), f^*(\boldsymbol{x}_{n+1})]U(\boldsymbol{x}, \boldsymbol{x}_{n+1}) \end{gathered} \tag{5}$$

where $\boldsymbol{r}(f,\ f^*)$ is a monotonous function toward variable f ($\boldsymbol{r}(f,\ f^*) \equiv 0$ if $f = f^*$). For this case we may assume that $\boldsymbol{r}(f,\ f^*)$ represents the difference between the two functions $f(\boldsymbol{x}_{n+1}) - f^*(\boldsymbol{x}_{n+1})$ which can be calculated during the training process. Hence

$$\boldsymbol{r}(f,\ f^*) = \eta(f^*,\ f)$$

$\eta(q)$ is a monotonously increasing function which satisfies the following condition

$$\eta(q) = \begin{cases} \le 0, \text{if } q \le 0; \\ > 0, \text{if } q > 0. \end{cases}$$

We suggest two different approaches in developing the function approximating algorithms based on potential functions (5). The first approach is expressed in the following way

$$f_{n+1}(\boldsymbol{x}) = f_n(\boldsymbol{x}) + \gamma_{n+1}\eta[f^*(\boldsymbol{x}_{n+1}) - f(\boldsymbol{x}_{n+1})] \quad (6)$$

Different implementations of algorithms (6) depend on the choice of γ and η. In this research we use the following modification of (6)

$$f_{n+1}(\boldsymbol{x}) = f_n(\boldsymbol{x}) + \gamma_{n+1}\, sign \\ [f^*(\boldsymbol{x}_{n+1}) - f(\boldsymbol{x}_{n+1})]U(\boldsymbol{x}, \boldsymbol{x}_{n+1}) \quad (7)$$

where γ_i is a non-negative numeric sequence satisfying the conditions discussed in the next section.

The second approach uses

$$f_{n+1}(\boldsymbol{x}) = f_n(\boldsymbol{x}) + \frac{1}{\Lambda}[f^*(\boldsymbol{x}_{n+1}) - f(\boldsymbol{x}_{n+1})]U(\boldsymbol{x}, \boldsymbol{x}_{n+1}) \quad (8)$$

where Λ is a positive constant satisfying condition

$\Lambda > \frac{1}{2} \sup_{x} U(\boldsymbol{x}, \boldsymbol{x}_{n+1})$.

We can use the approach for building the potential functions developed in [26, 27]. In this case the potential function for any input vector $\boldsymbol{x}_k$ is defined by the expression

$$\mathrm{U}(\boldsymbol{x}, \boldsymbol{x}_k) = \sum_{k=0}^{\infty} \lambda_i^{\,2} \varphi_i(\boldsymbol{x})\varphi_i(\boldsymbol{x}_k) \quad (9)$$

or

$$U(\boldsymbol{x}) = \sum_{i=1}^{\infty} w_i \varphi_i(\boldsymbol{x}) \quad (10)$$

where $\varphi_i(\boldsymbol{x})$, i = 1, 2, …are orthonormal functions; λ_i, i = 1, 2, … are real numbers for scaling different from zero; the weight w_i are unknown and can be determined iteratively from the input vectors [26, 27].

By letting

$$\psi_i(\boldsymbol{x}) = \lambda_i \varphi_i(\boldsymbol{x}) \quad (11)$$

the potential function in (9) becomes

$$\mathrm{U}(\boldsymbol{x}, \boldsymbol{x}_k) = \sum_{k=0}^{\infty} \psi_i(\boldsymbol{x})\psi_i(\boldsymbol{x}_k) \equiv (\psi_i(\boldsymbol{x})\psi_i(\boldsymbol{x}_k)) \quad (12)$$

We assume that

$$\mathrm{U}(\boldsymbol{x}, \boldsymbol{x}_k) = \sum_{k=0}^{\infty} \psi_i(\boldsymbol{x})\psi_i(\boldsymbol{x}_k) \equiv (\psi_i(\boldsymbol{x})\psi_i(\boldsymbol{x}_k)) \le M \quad (13)$$

where M is independent from $\boldsymbol{x}$ constant.

Therefore if we take into account (12) formulas (7) and (8) can be expressed in the following way

$$\boldsymbol{w}_{n+1}^{i} = \boldsymbol{w}_{n}^{i} + \gamma_n\, sign[f^*(\boldsymbol{x}_{n+1}) - \\ \sum_{i=1}^{\infty} \boldsymbol{w}_{n}^{i} \psi^i(\boldsymbol{x}_{n+1})]\ \psi^i(\boldsymbol{x}_{n+1}) \quad (14)$$

$$\boldsymbol{w}_{n+1}^{i} = \boldsymbol{w}_{n}^{i} + \frac{1}{\Lambda}[f^*(\boldsymbol{x}_{n+1}) - \\ \sum_{i=1}^{\infty} \boldsymbol{w}_{n}^{i} \psi^i(\boldsymbol{x}_{n+1})]\ \psi^i(\boldsymbol{x}_{n+1}) \quad (15)$$

The latter represent one possible way for neural network weight update in solving the problem of function approximation using potential functions built on orthonormal functions.

IV. Function approximation with symmetrical potential functions

The proposed neural network is based on RBFs with symmetrical potential functions and has two fundamental components – potential function generators (PFG) and potential function entities (PFE). It uses a two-step data processing structure:

a) A nonlinear transformation the input data undergoes via the PFGs in the network hidden layer;
b) A linear combination of basis functions responses to provide the network output through PFE.

The presented neural network learns by allocating new units and adjusting the parameters of the neural network based on the novelty of the presented sample. The proposed NN is able to control the complexity of its structure by allocating a new hidden unit to correct its response to the presented pattern. Fixed-size networks either use too few units or too many in which case the network demonstrates poor learning or poor generalization. The newly allocated units do not interfere with previously allocated units.

The placement and parameterization of the local units as well as the choice of their number is the difficult and critical part with RBFs networks. Networks with too many parameters can overfit data and have poor generalization. The method we present, however, allows effective determination of all these values automatically. The proposed approach is suitable for on-line and off-line applications.

Let us denote by $U(\boldsymbol{x}, \boldsymbol{x}_j)$ the potential function, which is connected to the learning pattern $\boldsymbol{x}_j \in \aleph, j = 1,\ 2,...,\ M$. Then, the appearance of training patterns corresponds to the generation of potential functions $U(\boldsymbol{x}, \boldsymbol{x}_1), U(\boldsymbol{x}, \boldsymbol{x}_2), ..., U(\boldsymbol{x}, \boldsymbol{x}_M)$, which is defined over the space $\aleph$.

The relation between the potential function $U(\boldsymbol{x},\boldsymbol{x}_n)$ and approximating function $f^*(\boldsymbol{x})$ is given by the next equation, which determines the iterative updates of the estimations $f_n(\boldsymbol{x})$.

$$f_{n+1}(\boldsymbol{x}) = q_n f_n(\boldsymbol{x}) + r_n U(\boldsymbol{x},\boldsymbol{x}_{n+1}) \tag{16}$$

where q_n, and r_n are numeric sequences which depend on one or all of the following factors: number of iteration n, the value of the current estimation f_n for the next input $\boldsymbol{x}_{n+1}$ ($f_n(\boldsymbol{x}_{n+1})$) and on the value of approximating function f^* for $\boldsymbol{x}_{n+1}$ - $f^*(\boldsymbol{x}_{n+1})$, if it is available. For our considerations we set $q_n \equiv 1$ and choose the following form for r_n:

$$r_n \equiv \gamma_n \left\{ F(f_n(\boldsymbol{x}_{n+1}), f^*(\boldsymbol{x}_{n+1})) + \mu_{n+1} \right\} \tag{17}$$

where $F(f_n, f^*)$ is a function of two variables; μ_n is a noise which appears during the measurement of $f^*(\boldsymbol{x})$ and γ_n is a non-negative numeric sequence satisfying the following conditions:

$$\sum_{n=1}^{\propto}\gamma_n = \propto;\ \gamma_n = const;\ \lim_{n\to\propto}\gamma_n = 0;\ \sum_{n=1}^{\propto}\gamma_n = \propto;\ \sum_{n=1}^{\propto}\gamma_n^2 = \propto;$$

The function $F(f, f^*)$ satisfies the next two conditions:

$$F(f^*, f^*) \equiv 0 \tag{18}$$

$$F(f,f^*)\begin{cases} \le 0, & if\ f \ge f^* \\ \ge 0, & if\ f \ge f^* \end{cases} \tag{19}$$

Thus, the sign of $f_{n+1}(\boldsymbol{x}_{n+1})$ - $f_n(\boldsymbol{x}_{n+1})$ might be determined by $F(f_n(\boldsymbol{x}_{n+1}), f^*(\boldsymbol{x}_{n+1}))$ as $U(\boldsymbol{x}_{n+1},\boldsymbol{x}_{n+1})$ and $\gamma_n > 0$. With the above assumptions if $f_n < f^*$, then $f_{n+1} > f_n$, which confirms that with the presentation of training pattern $\boldsymbol{x}_{n+1}$ to the neural network, the approximating functions change towards the function f^*. The same is valid for the case of $f_n > f^*$.

The weight update takes the following form:

$$w_{n+1}^i = q_n w_n^i + r_n \lambda_i^2 \varphi_i(\boldsymbol{x}_{n+1}) \tag{20}$$

Let $\psi_i(\boldsymbol{x}) = \lambda_i\varphi_i(\boldsymbol{x})$ and $w_n^i = w_n^i / \lambda_i$. Then

$$f_n(\boldsymbol{x}) = \sum_i w_n^i \varphi_i(\boldsymbol{x}) = \sum_i \tilde{w}_n^i \psi_i(\boldsymbol{x}) \tag{21}$$

Thus (20) takes the form:

$$\tilde{}w_{n+1}^i = q_n \tilde{}w_n^i + r_n\psi_i(\boldsymbol{x}_{n+1}) \tag{22}$$

where $\sum_{i=1}^{\propto}\lambda_i^2 < 0$ and $\lambda_i \ne 0$.

The case of supervised learning assumes a prior knowledge about the value of function $f^*(\boldsymbol{x}_{n+1})$ for the training pattern $\boldsymbol{x}_{n+1}$. As a result, the approximating function can be represented in the form:

$$\begin{aligned} f_{n+1}(\boldsymbol{x}) &= f_n(\boldsymbol{x}) + \gamma_{n+1} \\ &[f^*(\boldsymbol{x}_{n+1}) - f_n(\boldsymbol{x}_{n+1})]U(\boldsymbol{x},\boldsymbol{x}_{n+1}) \end{aligned} \tag{23}$$

where $\sum_{n=1}^{\propto}\gamma_n$ is a numeric sequence satisfying the conditions?

a) $\sum_{n=1}^{\propto}\gamma_n^2$ does not converge;

b) $f_n^*(\boldsymbol{x}_{n+1})$ converges.

The weight update under the above assumption determined by (20) and (22) is obtainable from

$$\begin{aligned} w_{n+1}^k &= w_n^k + \gamma_{n+1}\ sign \\ &[f^*(\boldsymbol{x}_{n+1}) - \sum_{n=1}^{N} w_n^k\ \varphi^k(\boldsymbol{x}_{n+1})]\quad \varphi^k(\boldsymbol{x}_{n+1}) \end{aligned} \tag{24}$$

V. Design concepts

Our approach is based on RBFNs with symmetrical potential functions (SPF), which perform a mapping based on a set of generated potential fields over the domain of input space by a number of potential function entities.

A given space $\aleph$ is symmetrical if a function $\rho(x,\ y)$ defined as a distance between any two points $x,\ y \in \aleph$ in the space, satisfies the following axioms:

- $\rho(x,\ y) = \rho(y,\ x)$;
- $\rho(x,\ y) = 0$, if and only if x coincide with y;
- $\rho(x,\ y) \ge 0$;
- Any three points x, y and z satisfy the triangle inequality
    - $\rho(x,y) \le \rho(x,z) + \rho(z,y)$
- Let $\boldsymbol{A}$ be a transformation, which assigns a point Ax to point x. Then, for each pair $\boldsymbol{x} \in X, \boldsymbol{y} \in Y$, the following condition is satisfied
    - $\rho(Ax, Ay) = \rho(x,y)$
- For any two points $x^{(1)}, y^{(1)}$ and $x^{(2)}, y^{(2)}$ that are at equal distance $\rho(x^{(1)},y^{(1)}) = \rho(x^{(2)},y^{(2)})$, there exists a

transformation such that the following conditions are met $x^{(2)} = Ax^{(1)}, y^{(2)} = Ay^{(1)}$.

Functions of two variables $\boldsymbol{U}(x, x_{\boldsymbol{k}})$, which fulfill the above conditions, and are mathematically expandable in infinite series, and satisfy the condition, $U(\boldsymbol{x}, \boldsymbol{x_k}) = U(\boldsymbol{x_k}, \boldsymbol{x})$ will be referred to as SPFs. Examples of such functions are given below by Eqs. (25)- (27).

$$U(\boldsymbol{x}, \boldsymbol{x_k}) = \exp\{-v\|\boldsymbol{x} - \boldsymbol{x_k}\|^2\} \tag{25}$$

$$U(\boldsymbol{x}, \boldsymbol{x_k}) = \frac{1}{1 + v\|\boldsymbol{x} - \boldsymbol{x_k}\|^2} \tag{26}$$

$$U(\boldsymbol{x}, \boldsymbol{x_k}) = \left|\frac{\sin v\|\boldsymbol{x} - \boldsymbol{x_k}\|^2}{v\|\boldsymbol{x} - \boldsymbol{x_k}\|^2}\right| \tag{27}$$

where v is a positive constant and $\|\boldsymbol{x} - \boldsymbol{x_k}\|$ is the norm of the vector $(\boldsymbol{x} - \boldsymbol{x_k})$. It is apparent that these functions are inversely proportional to the squared distance measure $D^2 = \|\boldsymbol{x} - \boldsymbol{x_k}\|^2$, which is also a characteristic, for example, of the force in a gravitational potential field.

A. Learning algorithm

We require class K_i to generate a higher accumulated potential for the input patterns in it than all other classes to all the samples in K_i. The PFUGNN2 network begins with no hidden units in the second layer. The observations are received sequentially during the learning, which is preferable compared to batch learning algorithms. The learning algorithm is described below.

Initialization:

PFUGNN2 is a three-layer network where the output response to an input pattern x for each predefined class is a linear combination of the hidden unit responses, determined by

$$f^{(K_k)}(\boldsymbol{x}) = w_0^{(K_k)} + \sum_{\boldsymbol{x}^* \in \aleph^*} w_r^{(K_k)} U(\boldsymbol{x}, \boldsymbol{x}_r) \tag{28}$$

where $w_0^{(K_k)}$ is the bias class weight ($k = 1,2,...,l$) set during the initialization phase, $w_1^{(K_k)}, w_2^{(K_k)},...,$ are the weights corresponding to the class K_k which are adjusted during training and $\boldsymbol{x}^* \in \aleph^*$ is the training subset corresponding to the class K_k which changes the respective potential field (classification rule). $U(\boldsymbol{x}, \boldsymbol{x_k})$ represents potential function.

Presented is a training pattern $\boldsymbol{x}_t \in K_q$, $1 \leq q \leq N$ from the learning set to the first layer of potential function generators (PFGs). The function $U(\boldsymbol{x}, \boldsymbol{x}_t)$ created by this pattern will be the first hidden unit of added to the hidden layer 2. Update the hidden unit q of the third layer (of cumulative potentials) in the following way:

$$U_1^{(K_q)}(\boldsymbol{x}) = w_1^{(K_q)} U(\boldsymbol{x}, \boldsymbol{x}_t),$$

where $w_1^{(K_q)}$ is initially set to 1.

Update the potential function entity for the class K_q - PFE$^{(K_q)}$ with $U(\boldsymbol{x}, \boldsymbol{x}_t)$.

Update the output class separated function for class K_q in the following way:

$$f_1^{(K_q)}(\boldsymbol{x}) = w_0^{(K_q)} + w_1^{(K_q)} U_1^{(K_q)}(\boldsymbol{x}) \tag{29}$$

Learning:

For each training pattern $\boldsymbol{x}_v \in K^p, 0 < p \leq l$ of the learning set, where $\boldsymbol{x}_v \in X \subseteq \Re^l$, $v = 1, 2,...,$ and l is the total number of classes.

1. **Compute** the potential function $U(\boldsymbol{x}, \boldsymbol{x}_n)$ created by this pattern (PFGs).
2. **Calculate** all cumulative potentials for the current pattern $\boldsymbol{x}_v$.

$$\begin{aligned} &U_i^{(K_1)}(\boldsymbol{x} = \boldsymbol{x}_v), \\ &U_i^{(K_2)}(\boldsymbol{x} = \boldsymbol{x}_v),..., \ U_i^{(K_l)}(\boldsymbol{x} = \boldsymbol{x}_v) \end{aligned} \tag{30}$$

3. **Apply** the criterion for adding a neuron
If $U_i^{(K_p)}(\boldsymbol{x}_v) < U_i^{(K_r)}(\boldsymbol{x}_v)$ for some values of $r = 1,2,...,l, \ r \neq p$ (for example, $r_1, r_2,...,r_u$ then $\boldsymbol{x}_v \in \{\boldsymbol{x}_1^*, \boldsymbol{x}_2^*,..., \boldsymbol{x}_n^*,...\}$ (training subset $\{\boldsymbol{x}_1^*, \boldsymbol{x}_2^*,..., \boldsymbol{x}_n^*,...\}$ require changes in the decision functions)
4. **Allocate** a new hidden unit $U(\boldsymbol{x}, \boldsymbol{x}_v)$ in layer 2
5. **Adjust** the p-th and $r_1, r_2,...,r_u$ units (corresponding to classes K_p, $K_{r_1}, K_{r_2},...,K_{r_u}$) of cumulative potential layer following the formulas

$$\begin{aligned} &U_{n+1}^{(K_q)}(\boldsymbol{x}) = U_n^{(K_q)}(\boldsymbol{x}) + \gamma_{n+1} \\ &sign(f^*(\boldsymbol{x}_{n+1}) - \sum_{\boldsymbol{x}^* \in \aleph^*} w_n^{(K_q)} U_n^{(Kq)}(\boldsymbol{x} = \boldsymbol{x}_{n+1})) U(\boldsymbol{x}, \boldsymbol{x}_{n+1}) \end{aligned} \tag{31}$$

where $q = p, r_1, r_2,...,r_u$.

Equation (31) is equivalent to the equations (32).

$$\begin{aligned} &U_{n+1}^{(K_p)}(\boldsymbol{x}) = U_n^{(K_p)}(\boldsymbol{x}) + \gamma_{n+1} U(\boldsymbol{x}, \boldsymbol{x}_v); \\ &U_{n+1}^{(K_r)}(\boldsymbol{x}) = U_n^{(K_r)}(\boldsymbol{x}) - \gamma_{n+1} U(\boldsymbol{x}, \boldsymbol{x}_v), \\ &\text{for } r = r_1, r_2,...,r_u \ \text{ as defined above;} \\ &U_{n+1}^{(K_s)}(\boldsymbol{x}) = U_n^{(K_s)}(\boldsymbol{x}), \ \text{ for } \ s \neq r_1, r_2,...,r_u. \end{aligned} \tag{32}$$

6. Update the PFEs for classes K_p, $K_{r_1}, K_{r_2}, \ldots, K_{r_u}$ - $\mathrm{PFE}^{(K_p)}, \mathrm{PFE}^{(K_{r_1})}, \ldots, \mathrm{PFE}^{(K_{r_u})}$ with $U(\boldsymbol{x}, \boldsymbol{x}_v)$ and the output decision functions following (31).

VI. Experiments and Discussion

The potential function is connected to the learning pattern. Then, the appearance of training patterns corresponds to the generation of potential functions, which is defined over the space. The relation between the potential function and approximating function determines the iterative updates of the estimations.

The training pattern from the learning set to the first layer of potential function generators and created by this pattern will be the first hidden unit added to the next hidden layer. This pattern will update the hidden unit of the third layer (of cumulative potentials).

The proposed NN was applied to well known approximation benchmarks with and without a noise. Two of the simulations are presented below. Problem one is to approximate Hermite function

$$f_{hermite} = 1.1(1 - x + 2x^2)\exp(-1/2x^2) \qquad (33)$$

The training set includes 40 randomly selected points in [-4, +4] interval, and the test data consists of 80 uniformly distributed points from the same interval. The results of approximation in 3D are shown in Fig. 1. The simulation was run with the number of iteration equal to 520. For the second experiment, we assume that the training data for the target function (21) was corrupted by noise uniformly distributed in [0.05, 0.05] while the test set remains intact. The result demonstrated robustness of the neural network in the presence of noise.

For the second simulation, shown in Fig.2, the target function is described by

$$f(x,y) = x^3 + 0.3x^2 - 0.4x - y, \quad x, y \in [-1,1] \qquad (34)$$

The training set consists of 21 points, which are chosen by uniformly partitioning the domain [-1, 1] with grid size of 0.1. The set comprises 100 points uniformly randomly sampled from the same domain.

For the third experiment, illustrated in Fig.3, the target function is the sine function (*f(x)=sin(x)*) over the domain [-4, 4]. The training set consists of 80 points. The output of the network is shown in Fig.3. The approximation is almost exact with an error of 0.001.

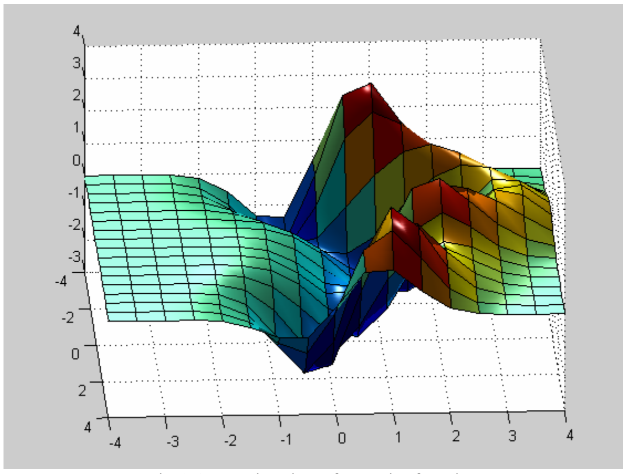

Fig.1 Approximation of Hermite function

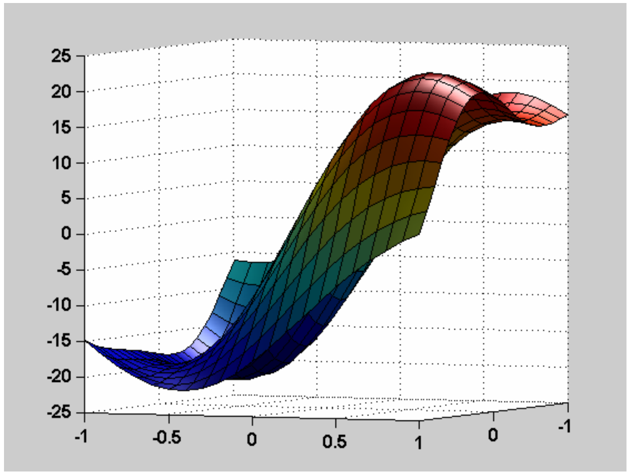

Fig.2 Simulation 2 results

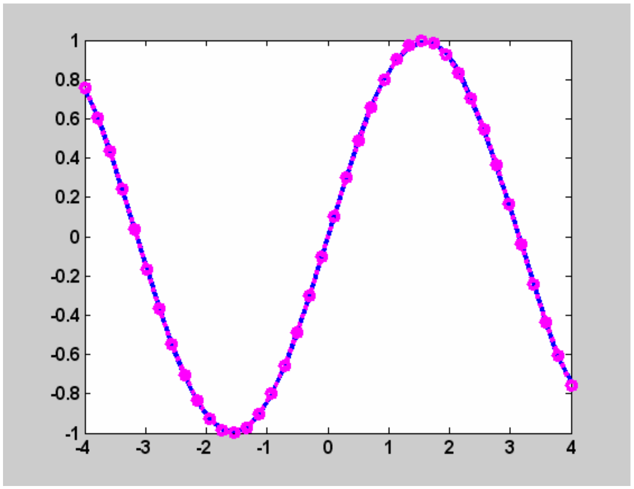

Fig.3 Simulation 2 results

VII. Conclusions

The determination of the type of RBFs used for a given set of data is an active area of research. The standard RBF neural network with a single output neuron produces a mapping function, where the dimensional input vector is submitted to the neural network and the scalar output is obtained to construct the classification rule.
The presented neural network learns by allocating new units and adjusting the parameters of the neural network based on the novelty of the presented sample.

All implementations show that the training algorithm is fast and straightforward, not only for noise-free nonlinear function approximation, but for the functions with noises as well.

Regardless of the proposed incremental feature of the neural network based on potential functions, pruning criterion can be developed by checking the values of potential functions during the training and creating additional rules on adaptation of cumulative potentials.

The NN is able to control the complexity of its structure by allocating a new hidden unit. The newly allocated units do not interfere with previously allocated units. The method we presented in this paper allows effective determination of all these values automatically.

References

[1] M. G. Lagoudakis and R. Parr. "Least-squares policy iteration". *Journal of Machine Learning Research*, 4(2003):1107–1149, 2003.

[2] *Handbook of Intelligent Control*, D. A. White and D. A. Sofge, Eds., Van Nostrand, New York, 1992, pp. 65–86. P. J. Werbos, Neurocontrol and Supervised Learning: An Overview and Evaluation.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning*. Cambridge, MA: MIT Press, 1998.

[4] T. Lyche, K. Mørken, and E. Quak, "Theory and algorithms for nonuniform spline wavelets," in *Multivariate Approximation and Applications*, N. Dyn, D. Leviatan, D. Levin, and A. Pinkus, Eds, Cambridge, U.K.: Cambridge Univ. Press, 2001.

[5] M. G. Cox, "Practical spline approximation," in *Lecture Notes in Mathematics965: Topics in Numerical Analysis*, P. R. Turner, Ed. New York: Springer-Verlag, 1982.

[6] A. Antoniadis and D. T. Pham, "Wavelets and statistics," in *LectureNotes in Statistics 103*. New York: Springer-Verlag, 1995.

[7] C. K. Chui, *An Introduction to Wavelets*. New York: Academic, 1992.

[8] J. H. Friedman, "Multivariate adaptive regression splines," *Ann. Statist.*, vol. 19, pp. 1–141, 1991.

[9] C. J. Stone, "The use of polynomial splines and their tensor products in multivariate function estimation," *Ann. Statist.*, vol. 22, pp. 118–184, 1994.

[10] M. G. Lagoudakis and R. Parr. "Least-squares policy iteration". *Journal of Machine Learning Research*, 4(Dec): 1107– 1149, 2003.

[11] F. Girosi, M. Jones, and T. Poggio. "Regularization theory and neural networks architectures". *Neural Computation*, 7(2):219–269, 1995.

[12] I. Daubechies. "Ten Lectures on Wavelets". *Society for Industrial and Applied Mathematics*, Philadelphia and Pennsylvania, 1992.

[13] Baldi P., Computing with Arrays of Bell-Shaped and Sigmoidal Functions. Eds. R.P. Lippman, J.E. Moody, and D.S.Touretzky, "Advances in Neural Network Information Processing Systems", Morgan-Kaufman, pp. 735-742, 1991.

[14] J.Park, I.Sandberg, "Approximation and Radial-Basis-Function Networks". *Neural computation*:5(3), pp.305-316, 1993.

[15] K. Hornik, M. Stinchcombe, and H. White, "Multi-layer feedforward networks are universal approximators," *Neural Networks.*, vol. 2, pp. 359–366, 1989.

[16] C. T. Chen and W. D. Chang, "A Feedforward Neural Network with Function Shape Autotunning". *Neural Networks*, 9(4), 627-641, 1996.

[17] L. Vecci, F. Piazza and A. Uncini, "Learning and Approximating Capabilities of Adaptive Spline Activation Function Neural Network". *Neural Networks*, 11, 259-270, 1998.

[18] K. Hornik, "Some New Results on Neural Network Approximation". *Neural Networks*, 6, 1069-1072, 1993.

[19] M. Leshno, V. Y. Lin, A. Pinkus and S. Schocken, "Multilayer Feedforward Networks with a Nonpolynomial Activation Function Can Approximate Any Function". *Neural Networks*, 6, 861-867, 1993.

[20] M. Zhang, J. Fulcher, and R. A. Scofield, Rainfall Estimation Using an Artificial Neural Network Group. *Neurocomputing*, 16, 97-115, 1997.

[21] Broomhead D. S. and Lowe D., "Multivariate Functional Interpolation and Adaptive Networks". *Complex Systems*, Vol. 2, pp. 321-355, 1988.

[22] Micchelli, C.A., "Interpolation of scattered data: Distance matrices and conditionally positive definite functions", *Constructive Approximation*, vol. 2, pp. 11–22, 1986

[23] Bors G., and Gbbouj M., "Minimal Topology for Radial Basis Functions Neural Networks for Pattern Classification". *Digital Processing*, vol. 4, pp. 173-188, 1994.

[24] S.Chen, C.Cowan, and P.Grant, "Orthogonal Least-Squares Learning Algorithms for Radial Basis Function Network". *IEEE Trans. Neural Networks*, vol.2, pp.302-309, 1991.

[25] I.Valova, N.Georgieva, R.Murat Demirer, P. Tchimev, and G. Georgiev, "The Determination of the Evoked Potential Generating Mechanism based on Radial Basis Neural Network Model", *Proceedings of the International Conference on Neural Networks*, Pennsylvania, 2000, pp. 24-28.

[26] N.Gueorguieva, I.Valova, "DYPOF: Dynamically Adaptive Neural Network with Potential Functions". *Journal of Smart Engineering System Design*, 5:517-536, 2003.

[27] N.Gueorguieva, I.Valova, "Building RBF Neural Network Topology through Potential Functions", *Springer-Verlag Series: Lecture Notes in Computer Science*. Vol. 2714. Artificial Neural Networks and Neural Information Processing, pp 1033 – 1040, Kaynak, E. Alpaydin, E. Oja, L. Xu (Eds.), ISBN 3-540-40408-2, 2003.